Searching Arrays

LAST	TODAY	NEXT
Arrays	Linear search	Cost
Safety	Correctness	Sorting arrays
	Contract failure	
	Contract exploits	

Array search

```
int search(int x, int[] A, int n)
{
 for (i = 0; i < n; i ++)
 {
  if (A[i] == x) return i;
 }
}
```

Is `A[i]` safe?

What is a reasonable precondition?

safe but ...

```
int search(int x, int[] A, int n)
//@requires n == \length(A);
{
 for (int i = 0; i < n; i ++)
  //@loop_invariant 0 <= i;
  {
    if (A[i] == x) return i;
  }
}
```

```
int search(int x, int[] A, int n)
//@requires n == \length(A);
{
 for (int i = 0; i < n; i ++)
 //@loop_invariant 0 <= i;
 {
   if (A[i] == x) return i;
 }
}
```

What do we return if x is not found?

```
int search(int x, int[] A, int n)
//@requires n == \length(A);
{
 for (int i = 0; i < n; i ++)
 //@loop_invariant 0 <= i;
 {
   if (A[i] == x) return i;
 }
 return   -1;
}
```

Client using search:

```
… //allocate B
int i =  search(12, B, 5);
if (i != -1) {
  B[i] = 13; //change 12 to 13
  …
}
```

How do we know `B[i]` is safe in the client's code?

The client sees only the contracts

```
int search(int x, int[] A, int n)
//@requires n == \length(A);
/*@ensures \result == -1 ||
          A[\result] == x);
  @*/
{
 for (int i = 0; i < n; i ++)
 //@loop_invariant 0 <= i;
 {
   if (A[i] == x) return i;
 }
 return -1;
}
```

What if `A[\result]` is unsafe?

```
int search(int x, int[] A, int n)
//@requires n == \length(A);
/*@ensures \result == -1 ||
          (0 <= \result && \result < n && A[\result] == x);
  @*/
{
 for (int i = 0; i < n; i ++)
 //@loop_invariant 0 <= i;
 {
   if (A[i] == x) return i;
 }
 return -1;
}
```

We rely on short-short-circuiting behavior of && to ensure the safety of A[\result]

What is wrong with this code?

```
int search(int x, int[] A, int n)
//@requires n == \length(A);
/*@ensures \result == -1 ||
          (0 <= \result && \result < n && A[\result] == x);
  @*/
{
  return -1;
}
```

Contract exploit!

A better contract for correctness

Return -1 only if x is not in **A[0,n)**

Example:

A

0	1	2	3	4	5
5	6	9	3	2	1

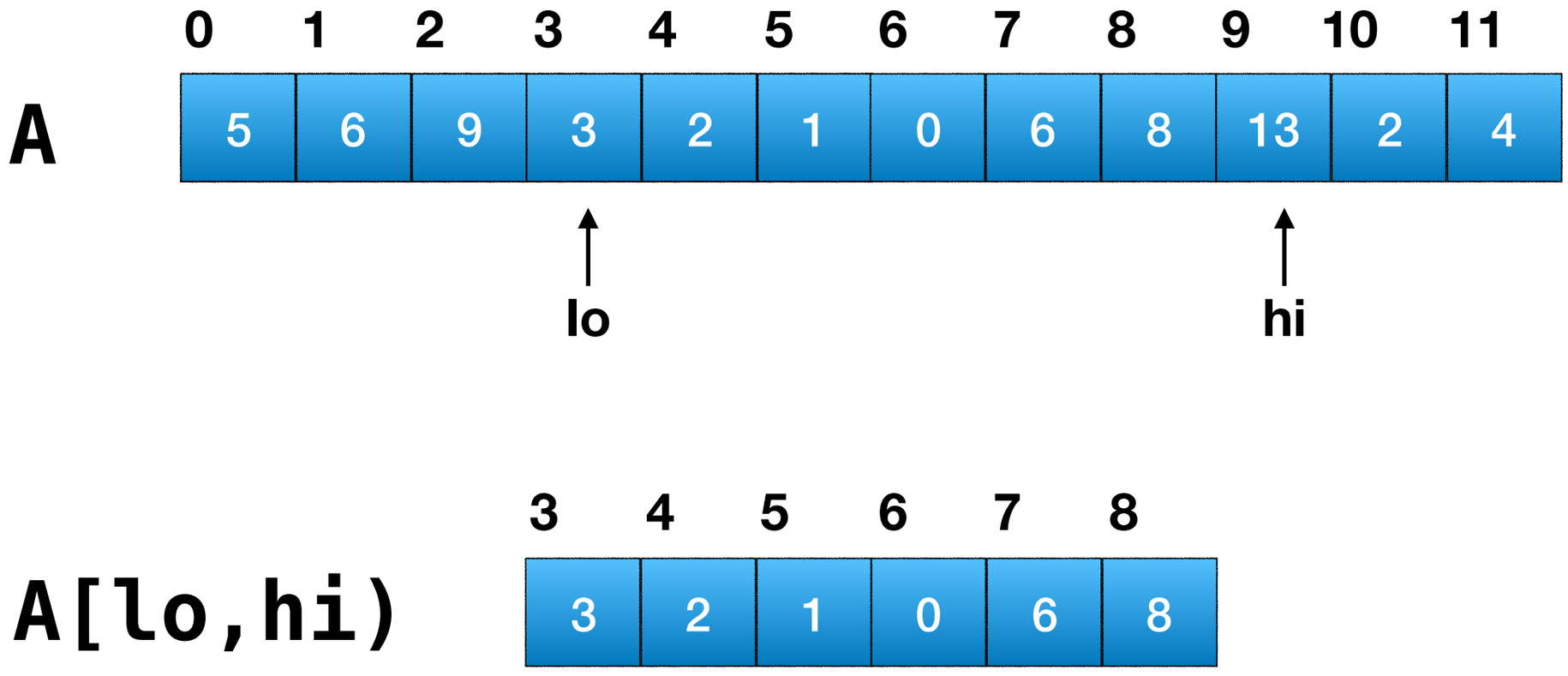
A[0,6)

0	1	2	3	4	5
5	6	9	3	2	1

A[0,3)

0	1	2
5	6	9

Array segments



lo == hi means an empty segment

x ∉ A[lo,hi)

```
bool is_in(int x, int[] A, int lo, int hi)
//@requires 0 <= lo && lo <= hi && hi == \length(A);
```

```
int search(int x, int[] A, int n)
//@requires n == \length(A);
/*@ensures (\result == -1 ______________________) ||
          (0 <= \result && \result < n && A[\result] == x);
  @*/

{



}
```

x ∉ A[0,n)

```
bool is_in(int x, int[] A, int lo, int hi)
//@requires 0 <= lo && lo <= hi && hi <= \length(A);
{
  if (lo == hi) return false;
  return A[lo] == x || is_in(x, A, lo+1, hi);
}


int search(int x, int[] A, int n)
//@requires n == \length(A);
/*@ensures (\result == -1 && !is_in(x, A, 0, n)) ||
          (0 <= \result && \result < n && A[\result] == x);
  @*/
{
  for (int i = 0; i < n; i++)
  //@loop_invariant 0 <= i;
  {
    if (A[i] == x) {
      return i;
    }
  }
  return -1;

}
```

Is the postcondition satisfied whenever the function returns?

Proving correctness

```
9 int search(int x, int[] A, int n)
10 //@requires n == \length(A);
11 /*@ensures (\result == -1 && !is_in(x, A, 0, n)) ||
12           (0 <= \result && \result < n && A[\result] == x);
13  @*/
14 {
15   for (int i = 0; i < n; i++)
16   //@loop_invariant 0 <= i;
17   {
18     if (A[i] == x) {
19       return i;
20     }
21   }
22   return -1;
23 }
```

return i

`0 <= i` (line 16, loop invariant)
`i < n` (line 15, loop guard)
`A[i] == x` (line 18, if condition)

return -1

How do we say logically that x is not in `A[0,n)`?
Introduce a loop invariant?

```
 9 int search(int x, int[] A, int n)
10 //@requires n == \length(A);
11 /*@ensures (\result == -1 && !is_in(x, A, 0, n)) ||
12           (0 <= \result && \result < n && A[\result] == x);
13  @*/
14 {
15   for (int i = 0; i < n; i++)
16   //@loop_invariant 0 <= i;
17   //@loop_invariant !is_in(x, A, 0, i);
18   {
19     if (A[i] == x) {
20       return i;
21     }
22   }
23   return -1;
24 }
```

x ∉ A[0,i)

INIT:

`!is_in(x, A, 0, 0)` (**A[0,0)** is empty)

PRES:

Assume `!is_in(x, A, 0, i)` **x ∉ A[0,i)**

Show `!is_in(x, A, 0, i')` **x ∉ A[0,i')**

`i' = i+1` (line 15, loop increment)

So, we need to show **x ∉ A[0,i+1),** which means

`!is_in(x, A, 0, i) && A[i] != x`

If `A[i] == x` (no proof obligation because we exit the loop)

Otherwise, immediate

Back to proving the postcondition

```
 9int search(int x, int[] A, int n)
10//@requires n == \length(A);
11/*@ensures (\result == -1 && !is_in(x, A, 0, n)) ||

12            (0 <= \result && \result < n && A[\result] == x);
13  @*/
14{
15  for (int i = 0; i < n; i++)
16  //@loop_invariant 0 <= i;
17  //@loop_invariant !is_in(x, A, 0, i);
18  {
19    if (A[i] == x) {
20      return i;
21    }
22  }
23  return -1;
24}
```

EXIT:

`!is_in(x, A, 0, i)` (line 17, LI)

`i >= n` (negation of the loop guard)

But we need i == n.

Add i <= n as a loop invariant?

```
 9 int search(int x, int[] A, int n)
10 //@requires n == \length(A);
11 /*@ensures (\result == -1 && !is_in(x, A, 0, n)) ||

12           (0 <= \result && \result < n && A[\result] == x);
13   @*/
14 {
15   for (int i = 0; i < n; i++)
16   //@loop_invariant 0 <= i && i <= n;
17   //@loop_invariant !is_in(x, A, 0, i);
18   {
19     if (A[i] == x) {
20       return i;
21     }
22   }
     //@assert !is_in(x, A, 0, n);
23   return -1;
24 }
```

EXIT:

`!is_in(x, A, 0, i)` (line 17, LI)

`i >= n` (negation of the loop guard)

`i <= n` (line 16, loop invariant)

i == n and !is_in(x, A, 0, n)

More contract exploits

```
int search(int x, int[] A, int n)
//@requires n == \length(A);
/*@ensures (\result == -1 && !is_in(x, A, 0, n)) ||
           (0 <= \result && \result < n && A[\result] == x);
  @*/
{
  for (int i = 0; i < n; i++)
  {
    A[i] == x; // modify A[i]
    return i;  // return i
  }
  return -1;
}
```

```
int search(int x, int[] A, int n)
//@requires n == \length(A);
/*@ensures (\result == -1 && !is_in(x, A, 0, n)) ||
          (0 <= \result && \result < n && A[\result] == x);
  @*/
{
  for (int i = 0; i < n; i++)
  {
    A[i] == x+1;
  }
  return -1;
}
```